3D Plant Phenotyping: All You Need is Labelled Point Cloud Data

Ayan Chaudhury[1,2], Frédéric Boudon[3,4], and Christophe Godin[1,2]

[1] INRIA Grenoble Rhône-Alpes, team MOSAIC
[2] Laboratoire Reproduction et Développement des Plantes, Univ Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRA
{ayan.chaudhury,christophe.godin}@inria.fr
[3] CIRAD, UMR AGAP, 34098 Montpellier, France
[4] AGAP, Univ Montpellier, CIRAD, INRAE, Institut Agro, Montpellier, France
frederic.boudon@cirad.fr

Abstract. In the realm of modern digital phenotyping technological advancements, the demand of annotated datasets is increasing for either training machine learning algorithms or evaluating 3D phenotyping systems. While a few 2D datasets have been proposed in the community in last few years, very little attention has been paid to the construction of annotated 3D point cloud datasets. There are several challenges associated with the creation of such annotated datasets. Acquiring the data requires instruments having good precision and accuracy levels. Reconstruction of full 3D model from multiple views is a challenging task considering plant architecture complexity and plasticity, as well as occlusion and missing data problems. In addition, manual annotation of the data is a cumbersome task that cannot easily be automated. In this context, the design of synthetic datasets can play an important role. In this paper, we propose an idea of automatic generation of synthetic point cloud data using virtual plant models. Our approach leverages the strength of the classical procedural approach (like L-systems) to generate the virtual models of plants, and then perform point sampling on the surface of the models. By applying stochasticity in the procedural model, we are able to generate large number of diverse plant models and the corresponding point cloud data in a fully automatic manner. The goal of this paper is to present a general strategy to generate annotated 3D point cloud datasets from virtual models. The code (along with some generated point cloud models) are available at: *https://gitlab.inria.fr/mosaic/publications/lpy2pc*.

Keywords: Procedural Model, L-System, Virtual Plant, Point Cloud, Labeled Synthetic Data

1 Introduction

With the recent breakthrough of the advancements on plant phenotyping research, botanical and agronomic studies have been revolutionized to enter in a

new stage of development [35]. In recent years, 3D computer vision based systems have been widely developed in different types of phenotyping applications including robotic branch pruning [5], automated growth analysis [8,7], agricultural automation tasks [38], etc. Point cloud based approach has been an integral part of machine learning aspects of plant phenotyping [47]. However, many of these techniques require ground truth data for training the algorithms and for validation of the results in a quantitative manner. For example, methods performing plant organ segmentation [27,22] require labeled point cloud data for quantifying the segmentation accuracy. Similarly, in-field 3D segmentation [34] or plant structure classification tasks [10] also demand annotated datasets for validation. In addition, to exploit the recent success of deep learning technologies for point cloud data [30] in agricultural applications, large amount of annotated data are in extreme demand. Unfortunately at the current moment, there is a scarcity of available annotated point cloud dataset in the plant phenotyping community. Only a handful of datasets of certain plant species are proposed in the literature. Among the currently available 3D datasets, Cruz *et al.* [9] created a multimodal database of top view depth images of Arabidopsis and bean plants. Wen *et al.* [44] proposed a database of 3D geometric plant models which are based on the measurements of in-situ morphological data. Different types of species in different growth stages are modelled in the framework. However the data is limited to the organ scale of the plants, and not on the full 3D model. Bernotas *et al.* [1] created a photometric stereo dataset along with manually annotated leaf masks of Arabidopsis rosettes. The Komatsuna plant dataset [37] contains RGB and depth images with annotated leaf labels. However, none of the above mentioned datasets are built on full 3D point cloud model of plants. Recently, the ROSE-X dataset [11] is created with 3D models of rosebush plants acquired through X-ray tomography method. Manual annotation is performed to label voxels and point clouds belonging to different organs of the plant. For remote sensing applications of vegetation studies, Wang [41] simulated a synthetic Terrestrial Laser Scanning (TLS) data of forest scenes. The generated point cloud contains simulated data of large trees having structurally complex crowns with labels belonging to leaf and wood class.

In contrast to the scarcity of 3D point cloud datasets, there has been quite a few 2D datasets available for public use, as well as software tool to access different types of existing solutions according to the experimental need [24]. These have revealed extremely useful for training machine learning algorithms and performance validations tasks for 2D image based plant phenotyping applications. The dataset proposed in [25] is one of the first kind, where annotated top-view color images of rosette plants at different growth stages are available. Shadrin *et al.* [31] proposed a manually annotated dataset of raw salad images acquired over a period of several days for plant growth dynamics assessment. Among other recent datasets, the Oil Radish dataset [26] and Grass clover dataset [32] have been proposed. In last few years, leaf segmentation and counting of rosette plants have been an active area of interest. Deep learning techniques have been very successful in solving this type of problem, which demand lot of annotated

data for training the network. In order to overcome the scarcity of available datasets, different types of data augmentation strategies have been proposed to generate realistic synthetic data. Giuffrida *et al.* [13] proposed a Generative Adversarial Network (GAN) model to artificially create Arabidopsis rosette images. The model allows the user to create plants with desired number of leaves in a realistic manner. With a similar type of motivation, Ward *et al.* [43] generated synthetic leaf images inspired by domain randomization strategy using a Mask-RCNN deep learning model. Kuznichov *et al.* [21] proposed a data augmentation strategy which preserves the geometric structure of the generated leaves with the real data. Different types of synthetic images are generated by applying some heuristic rules on the leaf shape and growth and recently, parametric L-systems have been used to perform data augmentation in generating synthetic leaf models [36]. Various leaf databases have also been proposed [33,46,20] which are useful for species identification and leaf architecture analysis [45,40,6].

Some of the difficulties associated with creating labeled 3D point cloud datasets include the expensive scanning devices, cumbersome and error prone manual annotation of large number of points, and reconstructing full 3D structure of complex plants in the presence of occlusion and noise. Also, growing varieties of plant species and acquiring data at different growth stages require lot of manpower and constant monitoring of the plants. In this regard, synthetic (or virtual) plant models can play an important role. Recently Ward *et al.* [42] demonstrated that synthetic data can be extremely useful in training deep learning models, which can be reliably used for measurements of different types of phenotypic traits in the general case. Modelling the geometry of complex plant structures have been a center of attention for mathematicians and biologists for decades [23,15,14,29]. Virtual models can be extremely useful in agricultural studies [12,3], as well as have great potential for mechanical simulation of plant behavior [4]. In generating the virtual plant models, L-system based modelling technique has been quite successful [28]. Different types of platforms have been developed to simulate the L-system rules, e.g., L+C modelling language [18], L-Py framework [2], etc. Functional Structural Plant Models (FSPM) [15] have emerged as powerful tool to construct 3D models of plant functioning and growth. These models can play an important role in mechanical simulation in crop science research [39]. Although generation of synthetic plant models have been a well studied area, the virtual plant models have not been exploited in generating synthetic 3D point cloud data for plant phenotyping applications. Compared to the synthetic artery data simulation software like Vascusynth [16] for medical imaging research, there is no such tool available in the plant phenotyping community.

Inspired by the necessity of synthetic plant point cloud datasets for 3D plant phenotyping applications, we propose a general approach to sample points on the surface of virtual plant models to generate annotated 3D point cloud data in a fully automatic manner. The framework is general in the sense that it can be plugged into any procedural plant model, and surface point cloud can be sampled from the virtual plant. Another motivation of the work comes from a

computer graphics perspective. Among different types of representations of geometric shapes in computer graphics, two types are widely used: triangular mesh and point based representation [19]. These two types are complementary to each other in terms of information richness and model complexity, and our framework builds a bridge between the two paradigms for the varieties of applications of the geometric plant models in phenotyping problems.

Initially we represent the virtual model as a collection of geometric primitives, each having their own label belonging to different organs of a plant. Each primitive is then tessellated into triangles to obtain a basic coarse mesh model. By a random sampling strategy using the idea of barycentric coordinates, we are able to generate uniformly distributed point cloud on the surface of the virtual model. Points residing inside any primitive are removed by a geometric filtering technique. We also model some basic characteristics of real world scanning devices by incorporating different sampling strategies, which can be adopted according to the need of the application. We demonstrate the efficiency of the approach by generating realistic synthetic point cloud model of Arabidopsis plants, as well as plants with different types of architecture. These datasets are generated by the stochasticity inherent to the procedural model and by a second level of stochasticity at the level of point cloud sampling. We also show some examples on other synthetic plants having different types of geometry structures.

In the next section we briefly describe how a virtual plant model should be specified to allow our algorithm to construct annotated point clouds from it. Then the basic random point cloud sampling strategy is explained, followed by the insideness testing process. Then we discuss the sampling strategy to obtain device specific point cloud models, and experimental results are demonstrated. Finally the scope of the work and directions for further improvements are discussed.

2 Virtual Plant Models

Virtual plants are computer generated synthetic models mimicking the development and growth of real plants. The models are typically developed by the recursive rules of L-systems [23]. One of the main attribute of L-system based plant models is that, complex plant structures can be generated by successive application of simple recursive rules. The recursive rules help in modelling natural phenomena like plant growth, creation and destruction of plant organs over time, change of shape (e.g. bending, spiralling), etc. This helps us to model a desired plant species in different growth stages by incorporating biological knowledge about the plant in the L-system rules.

In a L-system framework, a plant is defined by a string of symbols, called L-string, representing the different organs of the plant, A for Apex, I for internode, L for leaf for example. Each symbol is given a set of arguments describing variables attached to the corresponding organ. For example an internode can be modeled as a cylinder of height h and radius r, a leaf can be modeled as a polygon of size s, an apex can have an age t, etc. Square brackets are used

to delineate branches within the L-string. An initial L-string, called the axiom, defines the initial state of the plant, before growth starts. For example, the plant may initially be composed of an apex A(t). This is represented as:

```
Axiom: A(0)  # meaning that the L-string initially contains a
               single apex at time 0
```

A typical L-system model, then consists of a set of rules that make it possible to evolve the initial L-string. For instance the following rule typically says that at each time step *dt* the apex produces an internode I, a branching apex A and a leaf L, and an apical apex A:

```
A(t) --> I(0.01,0.1) [+(angle)[L(0.1)]A(t+dt)] A(t+dt)
```

where 'angle' is a variable (that can be a constant for example). This rule can be complemented by rules that express how internodes and leaf should change in time:

```
I(r,h) --> I(r',h')
    with r' = r + a*dt and h' = r' + b*dt
L(s) --> L(s')
    with s' = s + c*dt
```

In general, these so-called production rules do not describe how the symbols are associated with geometric models. For this, a way to interpret geometrically the symbols of the L-string should be defined. This is done using a new set of rules called interpretation rules. Here they could for instance look like, e.g.:

```
A(t) --> Sphere(radius(t))
    with radius(t) = 0.1 (constant radius in time)
I(r,h) --> Cylinder(r,h)
L(s) --> Polygon(size)
```

To use our algorithm, one would need to attach a unique annotation for each class of organ that one would like to recognize in the application. This is done by adding an extra 'label' argument to the symbols of interest.

```
A(t) --> I(0.01,0.1,8) [+(angle)[L(0.1,9)]A(t+dt)] A(t+dt)
I(r,h,label_I) --> I(r',h',Label_I)
    with r' = r + a*dt and h' = r' + b*dt
L(size,label_L) --> L(s',label_L)
    with s' = s + c*dt
```

where label=8 is attached to internodes and label=9 to leaves. The interpretation rules should be modified accordingly:

```
A(t) --> Sphere(radius(t))
    with radius(t) = 0.1 (constant radius in time)
I(r,h,label_I) --> Cylinder(r,h,label_I)
L(size,label_L) --> Polygon(size,label_L)
```

In this way, the geometric models will be tagged with a label corresponding to their category (here Internode or Leaf), and will make it possible to identify the triangles (which are used for rendering the model) that belong respectively to internodes and leaves in the 3D scene. Basically we transfer the label of an organ to the triangles while performing the tessellation, as discussed next.

3 Surface Point Cloud Generation

After creating the virtual model, we aim at generating points on the surface of the model where each point has a label of the plant organ it belongs to. In order to achieve this, we follow a series of steps. Given a parametric L-system model with unique label associated with each organ of the plant, we extract the set of primitives (along with their label of the organ they belong to in the procedural model) which are used to display the virtual plant model as a 3D object. In general, the following types of primitives are typically used in constructing the 3D object: cylinder, frustum, sphere, Beźier curve, NURBS patch, etc. Each primitive is then locally tessellated into a set of triangles (these triangles are used to render the 3D object), and the triangles are assigned the label of the primitive. Finally a global list of labelled triangle vertices are created to obtain a coarse mesh model of the 3D structure of the plant. Now the idea is to sample a dense set of labelled point cloud on the triangle surfaces, with a controlled density to create a realistic point cloud model of the plant. In the next section, we describe a simple and effective random point sampling strategy that can generate desired number of labelled points on the surface of the model.

3.1 Random Point Sampling

In order to generate labelled points on the surface of the virtual model, we propose two types of sampling strategies. The first type of sampling aims at generating uniform density of points on the model surface, and the second type of sampling is motivated by the effect of real world scanning devices, for which point density depends on various sensor-related parameters. In this section, we present the basic sampling strategy to generate point cloud of uniform density.

As described earlier, at this stage we represent the virtual model as a set of triangles of different sizes in a 3D scene. Sampling of points on the triangle surface is performed by random selection of triangle in each iteration and generating a point at random location inside the triangle. If we want to generate a point cloud of size m, we repeat the process of random triangle selection m times, and each time we generate a random point inside the selected triangle. However since the triangle sizes are not uniform, the part of the model having higher number of small triangles will likely to have higher point density than the part having smaller number of large triangles. In order to handle this problem, we use the triangle area as a selection probability while performing the sampling, thus resulting in a list of triangles sampled in a uniform manner.

Next, in order to perform point sampling on each triangle surface, we exploit barycentric coordinates, which are widely used in ray tracing applications of computer graphics. The barycentric coordinates can be used to define the position of any point located inside a triangle. In the general case for a simplex in an affine space, let the vertices are denoted as $(\mathbf{v}_1, \cdots, \mathbf{v}_n)$. If the following condition holds true for some point $\mathbf{P}$:

$$(\alpha_1, \cdots, \alpha_n)\mathbf{P} = \alpha_1\mathbf{v}_1 + \cdots + \alpha_n\mathbf{v}_n, \tag{1}$$

and at least one of the α_i's are not zero, then the coefficients $(\alpha_1, \cdots, \alpha_n)$ are the barycentric coordinates of $\mathbf{P}$ with respect to $(\mathbf{v}_1, \cdots, \mathbf{v}_n)$. For the case of triangles, any point inside a triangle can be described by a convex combination of its vertices. Let the vertices of a triangle are denoted as $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. Then any random point $\mathbf{Q}$ inside the triangle can be expressed as,

$$\mathbf{Q} = \sum_{i=1}^{3} \alpha_i \mathbf{v}_i, \tag{2}$$

where $\alpha_1, \alpha_2, \alpha_3 \geq 0$ are three scalars (the barycentric coordinates of the point $\mathbf{Q}$) such that the following condition holds true:

$$\alpha_1 + \alpha_2 + \alpha_3 = 1. \tag{3}$$

For all points inside the triangle, there exists a unique triple $(\alpha_1, \alpha_2, \alpha_3)$ such that the condition of Equation 2 holds true. Hence, due to the property in Equation 3, any point can be defined by any two of the barycentric coordinates.

Using barycentric coordinates, random points can be generated easily on the triangle surfaces. For each randomly selected triangle, we generate two random numbers α_1 and α_2. Then α_3 is computed as, $\alpha_3 = 1-(\alpha_1+\alpha_2)$ (using Equation 3). This gives us a random point $\mathbf{Q}$ inside the triangle. By performing this process for m number of times, a uniformly spaced point cloud of size m is obtained on the surface of the virtual model.

3.2 Insideness Testing

Although the point sampling is performed on the surface of the primitives, there can be points remaining inside the volumetric primitives (cylinder, frustum, and sphere). As shown in Figure 1, for the case of a simple branching structure approximated by cylinders and frustums, part of a branch remains inside the other. This is physically inconsistent, and we need to filter out the points inside any volumetric primitive. Hence for each point, we perform a geometric testing for different types of primitives as follows.

Cylinder Primitive Let an arbitraily oriented cylinder has radius r and two endpoints $(\overrightarrow{p_i}, \overrightarrow{p_j})$ on the central axis. Given a random point $\overrightarrow{q}$, we aim at determining if the point is inside the cylinder or not (left of Figure 2). First, we

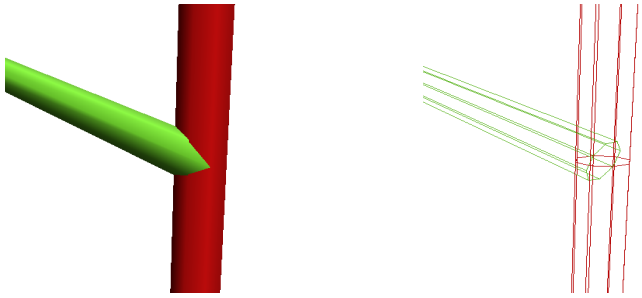

Fig. 1. In case of volumetric primitives, joining of two (or more) branches results in one primitive to retain inside another primitive.

test if $\overrightarrow{q}$ lies between the planes of the two circular facets of the cylinder. This is determined by the following two conditions:

$$\begin{aligned}(\overrightarrow{q} - \overrightarrow{p_i}) \cdot (\overrightarrow{p_j} - \overrightarrow{p_i}) \geq 0, \\ (\overrightarrow{q} - \overrightarrow{p_j}) \cdot (\overrightarrow{p_j} - \overrightarrow{p_i}) \leq 0.\end{aligned} \tag{4}$$

If the above conditions are true, we compute the perpendicular distance of the point $\overrightarrow{q}$ from the central axis of the cylinder. If the distance is less than the radius of the cylinder, then the point lies inside. Mathematically the condition is defined as,

$$\frac{|(\overrightarrow{q} - \overrightarrow{p_i}) \cdot (\overrightarrow{p_j} - \overrightarrow{p_i})|}{|(\overrightarrow{p_j} - \overrightarrow{p_i})|} \leq r. \tag{5}$$

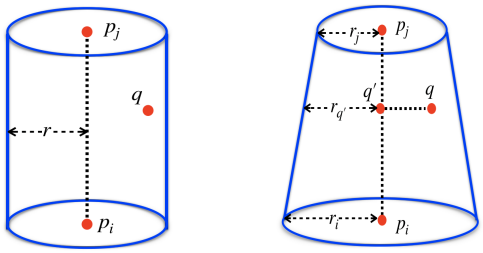

Fig. 2. Insideness testing for the case of cylinder and frustum.

Frustum Primitive Let an arbitrarily oriented frustum has base radius r_i and top radius r_j. The two endpoints on the central axis are defined as $(\overrightarrow{p_i}, \overrightarrow{p_j})$. Given a random point $\overrightarrow{q}$, we aim at determining if the point is inside the frustum or not (right of Figure 2).

Similar to the case of cylinder, first we check if the point lies between the planes of the two circular facets of the frustum using Equation 4. Then we project the point $\overrightarrow{q}$ on the line joining $\overrightarrow{p_i}$ and $\overrightarrow{p_j}$ (the central axis of the frustum) as,

$$\overrightarrow{q}' = \overrightarrow{p_i} + \left[\frac{(\overrightarrow{q} - \overrightarrow{p_i}) \cdot (\overrightarrow{p_j} - \overrightarrow{p_i})}{(\overrightarrow{p_j} - \overrightarrow{p_i}) \cdot (\overrightarrow{p_j} - \overrightarrow{p_i})}\right] * (\overrightarrow{p_j} - \overrightarrow{p_i}). \tag{6}$$

The distance between $\overrightarrow{q}$ and $\overrightarrow{q}'$ is computed as, $d = \text{dist}(\overrightarrow{q}, \overrightarrow{q}')$. Next we compute the frustum radius $r_{q'}$ at the point $\overrightarrow{q}'$. This can be computed as,

$$r_{q'} = \frac{\text{dist}(\overrightarrow{p_i}, \overrightarrow{q}')}{\text{dist}(\overrightarrow{p_i}, \overrightarrow{p_j})} * r_i + \frac{\text{dist}(\overrightarrow{p_j}, \overrightarrow{q}')}{\text{dist}(\overrightarrow{p_i}, \overrightarrow{p_j})} * r_j. \tag{7}$$

If the distance $d < r_{q'}$, then the point lies inside the frustum.

Sphere Primitive Let a sphere centered at point $\overrightarrow{p} = (x_c, y_c, z_c)$ having radius r. A point $\overrightarrow{q} = (x_i, y_i, z_i)$ is inside the sphere if the following condition holds true:

$$\sqrt{(x_c - x_i)^2 + (y_c - y_i)^2 + (z_c - z_i)^2} \leq r. \tag{8}$$

General Primitives For more general geometric models that represent more complex plant organs, we use general computer graphics techniques like ray casting [17] that make it possible to test whether a particular point lies inside or outside a closed geometric envelop.

The step by step procedure to generate the point cloud using the basic point sampling strategy is outlined in Algorithm 1.

3.3 Sensor Specific Distance Effect

The point cloud sampling described so far, is based on the assumption that the point density is uniform in the whole plant model. However in real scanning devices, this is usually not the case. While there are different types of sensor devices having different types of specific traits associated with the generated surface point cloud, one of the most common is the effect of distance from the scanner on the point density. A typical effect of this type yields higher point density near the scanner, and the density decreases as the distance of different parts of the object from the scanner increases. In order to simulate the effect of these type of real scanning devices, we implemented a sampling strategy which can be adapted to different types of cases according to the need of the application.

Instead of random sampling of triangles in every iteration as in the case of naive sampling technique described before, we compute the number of points to be sampled on each triangle based on a distance factor. First we compute the total surface area $\mathcal{A}_\mathcal{T}$ of all the triangles as, $\mathcal{A}_\mathcal{T} = \sum_i \mathcal{A}_{\mathcal{T}_i}$. Then we compute the point density ρ_u per unit area of the mesh as, $\rho_u = n/\mathcal{A}_\mathcal{T}$, where n is the maximum number of points we wish to sample. Basically ρ_u gives the point

density for the case of uniform sampling: given a random triangle $\mathcal{T}_i$, the number of points on the triangle surface can be approximated as $(\rho_u * \mathcal{A}_{\mathcal{T}_i})$.

Next we compute the distance d_i of each triangle centroid from the camera position, and normalize the value in the range $[0, 1]$. Now the idea is that, as the distance of a triangle increases from the camera position, the point density ρ_u is decreased by some function of d_i. For i-th triangle, the number of points to be sampled is computed as,

$$n_i = \rho_u \ \mathcal{A}_{\mathcal{T}_i} \ \gamma(d_i), \tag{9}$$

where $\gamma(\cdot)$ is a real valued function that controls the effect of distance on the point sampling. To demonstrate the effect of $\gamma(\cdot)$, we show an example of a simple branching structure in Figure 3, where we used $\gamma(d_i) = 1$ (which is the uniform sampling), and $\gamma(d_i) = d_i^2$. The Figure shows how the point density decreases as the distance of the object part increases from the camera position, compared to the case of uniform sampling.

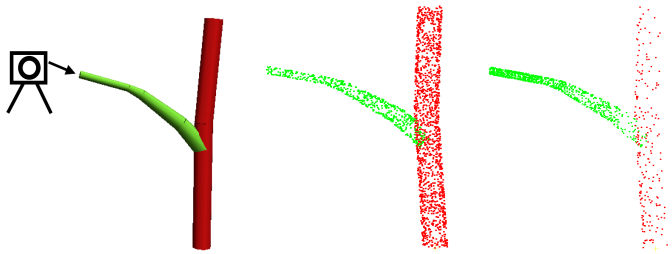

Fig. 3. Simulation of the effect of distance from the scanner position. From left to right: the original model along with camera position, using $\gamma(d_i) = 1$ (uniform point sampling), and $\gamma(d_i) = d_i^2$.

3.4 Effect of Noise

Presence of noise is a very common effect of scanning devices. While the type of noise can vary among different types of devices, we simulate a few common effects of noise in our framework. Some examples are shown in Figure 4 on a simple branching structure as in Figure 3. On the left of the figure, we show the effect of uniform noise, where each point is generated by a Gaussian having the surface position of the point as mean and a standard deviation of $\sigma = 0.06$. We also check the insideness testing while applying the noise, so that the noisy points do not remain inside any branch. In the middle of the figure, the noise is modelled with the effect of distance from the camera, as described in the previous section. Points closer to the scanner has almost zero noise, while the noise increases as the distance of the point increases from the position of the scanner. Finally at the right of the figure, the distance effect is shown considering the decrease of point density as described before.

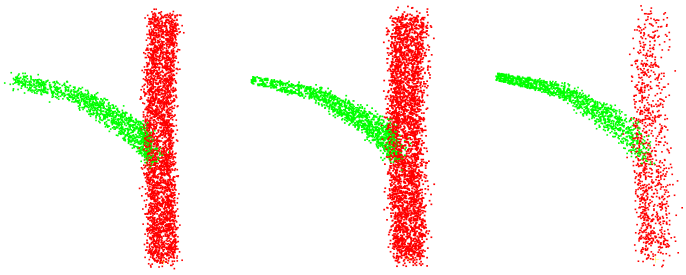

Fig. 4. Simulation of noise effect. Left: uniform distribution of noise, Middle: Effect of noise on the distance from the scanner, Right: Effect of noise on the distance along with point density factor.

4 Results

We have implemented the model in L-Py framework [2]. However, other types of frameworks (e.g. L+C [18]) can also be used. In the left of Figure 5, we show the synthetic model of an Arabidopsis plant that is generated by parametric L-system using stochastic parameters to compute different instances of the plant architecture. Using the stochasticity in the model, it is possible to generate a large variety of plants along with their labelled point cloud data. In the same figure we show some of the varieties of the generated point cloud models of the Arabidopsis plant. In all examples, we have used 3 labels (flowers, internodes and leaves) and about 100k points, although the number of labels and points can be controlled according to the need of the application. In Figure 6, we show more detailed result of the Arabidopsis plant.

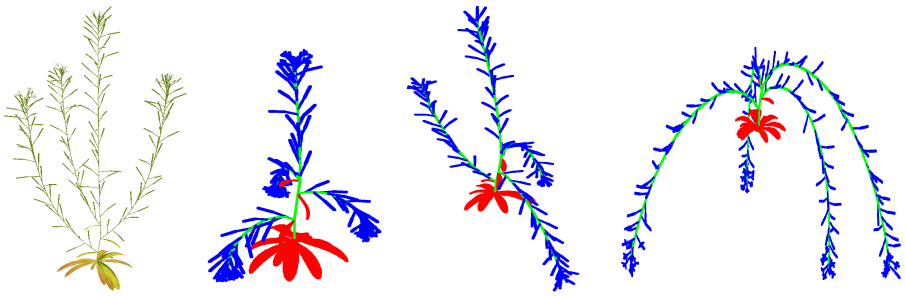

Fig. 5. From left to right: a virtual model of Arabidopsis plant, followed by labelled point cloud representation of the plant model in different growth stages.

We also show some results on different types of artificial plants in Figure 7. This demonstrates the portability of the framework which allows us to use it in any procedural plant model. In the last example (rightmost in the same figure),

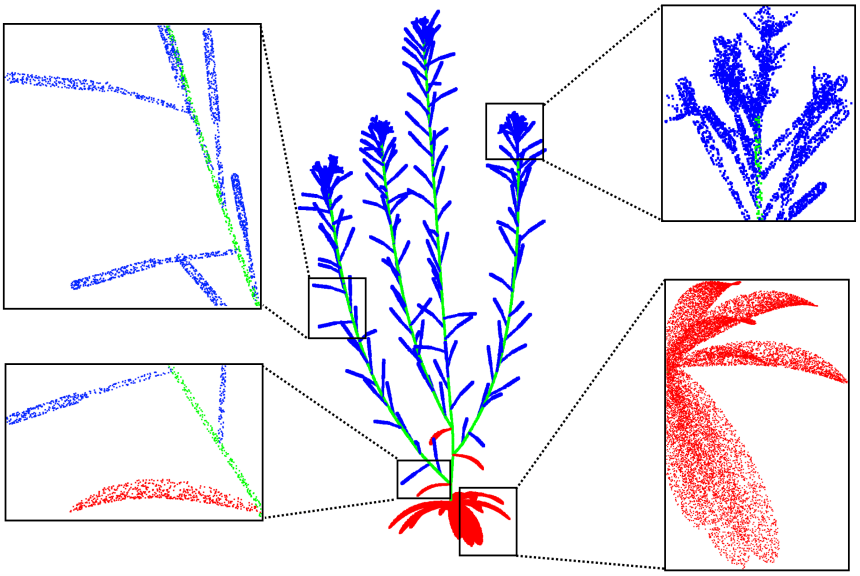

Fig. 6. An example of a point cloud model generated by the approach using 3 labels and about 100k points. Different parts are shown in details by the zoom-in.

we demonstrate that the number of labels can be controlled according to the need of the application. In this case, we have used different labels for each flower of the Lilac plant (denoted by different colors).

5 Discussion

There are many ways the framework can be improved and adapted to different types of applications. For example, incorporating back face culling in the framework can be useful for applications where occlusion needs to be modelled. While we have implemented a basic sampling strategy to mimic the sensor specific distance effect, there can be several other types of effects in real world scanning devices, that yield a research avenue by themselves. Incorporating these effects in the point sampler will make the system more versatile and robust, and increase its application range. As we have simulated the effect of distance on the point density by considering the distance of the triangle centroids from the scanner, the point density remains same in all areas of one triangle. For large triangles where one part of the triangle is closer to the scanner than the other part, ideally the point density should vary accordingly inside the same triangle. However in the current framework, we have not modelled this effect.

In order to model the sensor specific distance effect, the desired number of points should be significantly higher than the total number of triangles in the model. In the current implementation, we estimate the number of points

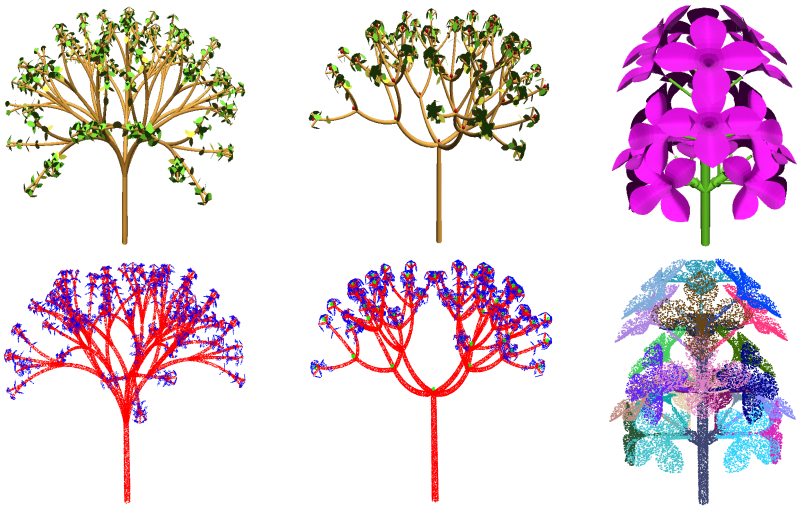

Fig. 7. Some examples of artificial plant models (top row), and the corresponding labelled point cloud data (bottom row). In the first two examples, 2 and 3 labels are used, respectively. In the last example, each flower is assigned a different label denoted by different colors.

per triangle based on its area, without taking into consideration the number of sampled points on the neighboring triangles. While the number of estimated points per triangle can be a fractional number according to Equation 9, we need to round off the value to a nearest integer. Hence the part(s) of the model having large number of tiny triangles will contain at least one point inside each triangle, even when the number of estimated point for the triangle is less than 1, resulting in higher point density in these areas. Ideally the sampling should be performed based on the point density of the neighboring triangles, although this is not considered in the framework.

The current implementation is not optimized, and thus is slow in terms of computation time. For example, in order to generate 100k points in the Arabidopsis model, it takes about 1.5 hours to run the basic sampler. While we have decomposed each shape into triangles to perform point sampling, the points can also be sampled directly on the surface of the geometrical shape. However, we use the triangle representation since it is simple and straightforward.

6 Conclusion

In this paper we have proposed a generalized approach to generate labeled 3D point cloud data from procedural plant models. As a preliminary work, we have presented results on a simulated Arabidopsis plant along with some artificial plant models. The point sampler is portable enough to use in different types of plant models, and corresponding point cloud data can be generated. This can be

beneficial to various types of plant phenotyping research problems in performing quantitative evaluation of point cloud based algorithms, as well as in training the machine learning models. It will be worth investigating how the synthetic models perform in training deep neural networks. In future, we plan to model different types of plant species and more complex plants in the framework. While currently there is no such method available to generate synthetic labeled point cloud data, our approach is the first step towards this in promoting 3D point cloud based plant phenotyping research.

Algorithm 1: Basic Point Sampling Algorithm

Input: Procedural Plant Model
Output: Surface point cloud of the model $\mathcal{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_n\}$, label list $\boldsymbol{\ell}$
begin
  $\mathcal{P} \leftarrow \{\Phi\}, \boldsymbol{\ell} \leftarrow \{\Phi\}$
  Assign label $\boldsymbol{\ell}_i, i \in \{1, \cdots, r\}$ to r organs in the procedural model
  Extract all primitives with organ labels from the model $\{\mathcal{C}_1^{\boldsymbol{\ell}_i}, \cdots, \mathcal{C}_m^{\boldsymbol{\ell}_i}\}$
  Extract the set of geometry parameters of the primitives $\{\lambda_1, \cdots, \lambda_m\}$
  Tessellate each primitive into set of triangles $\mathcal{T} = \{\{\mathcal{T}_1^{\boldsymbol{\ell}_i}\}, \cdots, \{\mathcal{T}_m^{\boldsymbol{\ell}_i}\}\}$
  Create global list of triangle vertices along with their corresponding labels
  while *Total number of Points* $!= n$ **do**
    Randomly select a triangle $\mathcal{T}_j^{\boldsymbol{\ell}_i} \in \mathcal{T}$ with area probability
    $\{\mathbf{v}_1^j, \mathbf{v}_2^j, \mathbf{v}_3^j\} \leftarrow vertex(\mathcal{T}_j^{\boldsymbol{\ell}_i})$ `// Extract triangle vertices`
    $\alpha_1 \leftarrow rand(0,1), \alpha_2 \leftarrow rand(0,1), \alpha_3 \leftarrow rand(0,1)$
    $\alpha_1 \leftarrow \alpha_1 / \sum_{i=1}^{3} \alpha_i, \alpha_2 \leftarrow \alpha_2 / \sum_{i=1}^{3} \alpha_i, \alpha_3 \leftarrow \alpha_3 / \sum_{i=1}^{3} \alpha_i$
    $\mathbf{p} \leftarrow \alpha_1 \mathbf{v}_1^j + \alpha_2 \mathbf{v}_2^j + \alpha_3 \mathbf{v}_3^j$
    insidenessFlag $\leftarrow False$
    `// Loop over all primitives`
    for $k \leftarrow 1$ **to** m **do**
      if $\mathcal{C}_k^{\boldsymbol{\ell}_i}$ *is a Cylinder* **then**
        insidenessFlag $\leftarrow$ isInsideCylinder$(\lambda_k, \mathbf{p})$`// using Eqn.4,5`
      else if $\mathcal{C}_k^{\boldsymbol{\ell}_i}$ *is a Frustum* **then**
        insidenessFlag $\leftarrow$ isInsideFrsutum$(\lambda_k, \mathbf{p})$`// using Eqn.4,6,7`
      else if $\mathcal{C}_k^{\boldsymbol{\ell}_i}$ *is a Sphere* **then**
        insidenessFlag $\leftarrow$ isInsideSphere$(\lambda_k, \mathbf{p})$`// using Eqn.8`
    if *insidenessFlag* $=$ *False* **then**
      $\mathcal{P} \leftarrow \mathcal{P} \cup \mathbf{p}$
      $\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} \cup \boldsymbol{\ell}_i$

Acknowledgment

This work is supported by Robotics for Microfarms (ROMI) European project.

References

1. Bernotas, G., et al.: A photometric stereo-based 3d imaging system using computer vision and deep learning for tracking plant growth. GigaScience **8**(5) (2019)
2. Boudon, F., Pradal, C., Cokelaer, T., Prusinkiewicz, P., Godin, C.: L-py: an l-system simulation framework for modeling plant architecture development based on a dynamic language. Frontiers in plant science **3**, 76 (2012)
3. Buck-Sorlin, G., Delaire, M.: Meeting present and future challenges in sustainable horticulture using virtual plants. Frontiers in plant science **4**, 443 (2013)
4. Bucksch, A., et al.: Morphological plant modeling: unleashing geometric and topological potential within the plant sciences. Frontiers in plant science **8**, 900 (2017)
5. Chattopadhyay, S., Akbar, S.A., Elfiky, N.M., Medeiros, H., Kak, A.: Measuring and modeling apple trees using time-of-flight data for automation of dormant pruning applications. In: IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 1–9 (2016)
6. Chaudhury, A., Barron, J.L.: Plant species identification from occluded leaf images. IEEE/ACM transactions on computational biology and bioinformatics (2018)
7. Chaudhury, A., Ward, C., Talasaz, A., Ivanov, A.G., Brophy, M., Grodzinski, B., Hüner, N.P.A., Patel, R.V., Barron, J.L.: Machine vision system for 3d plant phenotyping. IEEE/ACM Transactions on Computational Biology and Bioinformatics **16**(6), 2009–2022 (2019)
8. Chaudhury, A., Ward, C., Talasaz, A., Ivanov, A.G., Hüner, N.P.A., Grodzinski, B., Patel, R.V., Barron, J.L.: Computer vision based autonomous robotic system for 3d plant growth measurement. In: 12th Conference on Computer and Robot Vision (CRV). pp. 290–296 (2015)
9. Cruz, J., Yin, X., Liu, X., Imran, S., Morris, D., Kramer, D., Chen, J.: Multi-modality imagery database for plant phenotyping. Machine Vision and Applications **7**, 1–15 (2016)
10. Dey, D., Mummert, L., Sukthankar, R.: Classification of plant structures from uncalibrated image sequences. In: 2012 IEEE Workshop on the Applications of Computer Vision (WACV). pp. 329–336 (2012)
11. Dutagaci, H., Rasti, P., Galopin, G., Rousseau, D.: Rose-x: an annotated data set for evaluation of 3d plant organ segmentation methods. Plant methods **16**(1), 1–14 (2020)
12. Evers, J.B., Vos, J.: Modeling branching in cereals. Frontiers in plant science **4**, 399 (2013)
13. Giuffrida, M.V., Scharr, H., Tsaftaris, S.A.: Arigan: Synthetic arabidopsis plants using generative adversarial network. In: Proceedings of ICCV Workshop on Computer Vision Problems in Plant Phenotyping. pp. 2064–2071 (2017)
14. Godin, C., Costes, E., Sinoquet, H.: Plant architecture modelling - virtual plants and complex systems. In: Turnbull, C.G.N. (ed.) Plant architecture and its manipulation, chap. 9. Blackwell Publishing (2005)
15. Godin, C., Sinoquet, H.: Functional–structural plant modelling. New phytologist **166**(3), 705–708 (2005)
16. Hamarneh, G., Jassi, P.: Vascusynth: Simulating vascular trees for generating volumetric image data with ground-truth segmentation and tree analysis. Computerized medical imaging and graphics **34**(8), 605–616 (2010)
17. Horvat, D., Zalik, B.: Ray-casting point-in-polyhedron test. In: Proceedings of the CESCG 2012: The 16th Central European Seminar on Computer Graphics (2012)

18. Karwowski, R., Prusinkiewicz, P.: Design and implementation of the l+c modeling language. Electronic notes in theoretical computer science **86**(2), 134–152 (2003)
19. Kobbelt, L., Botsch, M.: A survey of point-based techniques in computer graphics. Computers & Graphics **28**(6), 801–814 (2004)
20. Kumar, N., Belhumeur, P.N., Biswas, A., Jacobs, D.W., Kress, W.J., Lopez, I., Soares, J.V.B.: Leafsnap: A computer vision system for automatic plant species identification. In: The 12th European Conference on Computer Vision (ECCV). pp. 502–516 (October 2012)
21. Kuznichov, D., Zvirin, A., Honen, Y., Kimmel, R.: Data augmentation for leaf segmentation and counting tasks in rosette plants. In: Proceedings of CVPR Workshop on Computer Vision Problems in Plant Phenotyping (2019)
22. Li, Y., Fan, X., Mitra, N.J., Chamovitz, D., Cohen-Or, D., Chen, B.: Analyzing growing plants from 4d point cloud data. ACM Transactions on Graphics (TOG) **32**(6), 1–10 (2013)
23. Lindenmayer, A., Prusinkiewicz, P.: The algorithmic beauty of plants, vol. 1. New York: Springer-Verlag (1990)
24. Lobet, G., Draye, X., Périlleux, C.: An online database for plant image analysis software tools. Plant methods **9**(1), 38 (2013)
25. Minervini, M., Fischbach, A., Scharr, H., Tsaftaris, S.A.: Finely-grained annotated datasets for image-based plant phenotyping. Pattern recognition letters **81**, 80–89 (2016)
26. Mortensen, A.K., Skovsen, S., Karstoft, H., Gislum, R.: The oil radish growth dataset for semantic segmentation and yield estimation. In: Proceedings of CVPR Workshop on Computer Vision Problems in Plant Phenotyping (2019)
27. Paulus, S., Dupuis, J., Mahlein, A., Kuhlmann, H.: Surface feature based classification of plant organs from 3d laserscanned point clouds for plant phenotyping. BMC bioinformatics **14**(1), 238 (2013)
28. Prusinkiewicz, P., Mündermann, L., Karwowski, R., Lane, B.: The use of positional information in the modeling of plants. In: Proceedings of SIGGRAPH. pp. 289–300 (2001)
29. Prusinkiewicz, P., Runions, A.: Computational models of plant development and form. New Phytologist **193**(3), 549–569 (2012)
30. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
31. Shadrin, D., Kulikov, V., Fedorov, M.: Instance segmentation for assessment of plant growth dynamics in artificial soilless conditions. In: Proceedings of BMVC Workshop on Computer Vision Problems in Plant Phenotyping (2018)
32. Skovsen, S., Dyrmann, M., Mortensen, A.K., Laursen, M.S., Gislum, R., Eriksen, J., Farkhani, S., Karstoft, H., Jorgensen, R.N.: The grassclover image dataset for semantic and hierarchical species understanding in agriculture. In: Proceedings of CVPR Workshop on Computer Vision Problems in Plant Phenotyping (2019)
33. Soderkvist, O.J.O.: Computer vision classification of leaves from swedish trees. Masters Thesis, Linkoping University, Sweden (2001)
34. Sodhi, P., Vijayarangan, S., Wettergreen, D.: In-field segmentation and identification of plant structures using 3d imaging. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 5180–5187 (2017)
35. Tardieu, F., Cabrera-Bosquet, L., Pridmore, T., Bennett, M.: Plant phenomics, from sensors to knowledge. Current Biology **27**(15), R770–R783 (2017)

36. Ubbens, J., Cieslak, M., Prusinkiewicz, P., Stavness, I.: The use of plant models in deep learning: an application to leaf counting in rosette plants. Plant methods **14**(1), 6 (2018)
37. Uchiyama, H., Sakurai, S., Mishima, M., Arita, D., Okayasu, T., Shimada, A., Taniguchi, R.: An easy-to-setup 3d phenotyping platform for komatsuna dataset. In: Proceedings of ICCV Workshop on Computer Vision Problems in Plant Phenotyping. pp. 2038–2045 (2017)
38. Vázquez-Arellano, M., Griepentrog, H.W., Reiser, D., Paraforos, D.S.: 3-d imaging systems for agricultural applications - a review. Sensors **16**(5) (2016)
39. Vos, J., Evers, J.B., Buck-Sorlin, J.H., Andrieu, B., Chelle, M., Visser, P.H.B.D.: Functional–structural plant modelling: a new versatile tool in crop science. Journal of experimental Botany **61**(8), 2101–2115 (2010)
40. Wang, B., Gao, Y., Sun, C., Blumenstein, M., Salle, L.J.: Can walking and measuring along chord bunches better describe leaf shapes? In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6119–6128 (2017)
41. Wang, D.: Unsupervised semantic and instance segmentation of forest point clouds. ISPRS Journal of Photogrammetry and Remote Sensing **165**, 86–97 (2020)
42. Ward, D., Moghadam, P.: Scalable learning for bridging the species gap in image-based plant phenotyping. Computer Vision and Image Understanding p. 103009 (2020)
43. Ward, D., Moghadam, P., Hudson, N.: Deep leaf segmentation using synthetic data. In: Proceedings of BMVC Workshop on Computer Vision Problems in Plant Phenotyping (2018)
44. Wen, W., Guo, X., Wang, Y., Zhao, C., Liao, W.: Constructing a three-dimensional resource database of plants using measured in situ morphological data. Applied Engineering in Agriculture **33**(6), 747–756 (2017)
45. Wilf, P., Zhang, S., Chikkerur, S., Little, S.A., Wing, S.L., Serre, T.: Computer vision cracks the leaf code. Proceedings of the National Academy of Sciences **113**(12), 3305–3310 (2016)
46. Wu, S.G., Bao, F.S., Xu, E.Y., Wang, Y., Chang, Y., Xiang, Q.: A leaf recognition algorithm for plant classification using probabilistic neural network. In: IEEE International Symposium on Signal Processing and Information Technology. pp. 11–16 (2007)
47. Ziamtsov, I., Navlakha, S.: Machine learning approaches to improve three basic plant phenotyping tasks using three-dimensional point clouds. Plant physiology **181**(4), 1425–1440 (2019)